Class and objects
Methods
Properties
Scope
Life Cycle
Autoloading
Interface and Inheritance
Composer and libraries

PHP
Object Oriented Programing
Advanced Programing

Class and Objects

The class and objects in PHP

- Class are blueprints for objects
- Class encapsulate shared elements
    - Called “static”
- Objects are created from class
- All objects are separated instance

The class and objects in PHP

- Class should have their own namespace
- Class should have their own file
- Class file should have the name of the class
- Class may not have more than 5 properties
- Class may not have more than 10 public methods
- Methods may not have more than 20 lines

The class and objects in PHP

- Class have block

- Class must be instantiated to create objects

```
namespace Human;

class Human
{
    // Properties / methods
}

$dude = new Human();
```

Methods

The class methods

- Methods have access scope
    - public
    - protected
    - private

- Methods can be static

The class methods

```
class Human
{
    public function askTo(Human $human, string
$what)
    {
        return $human->explain($what);
    }

    protected function explain($what)
    {
        if (!empty($what)) {
            return $this->thinkAbout($what);
        }
        return 'I don\’t understand';
    }

    private function thinkAbout($what)
    {
        return 'I don\'t care about this kind of thing';
    }
}
```

```
$dude = new Human();
$boss = new Human();

$boss->askTo($dude, 'Yoda vs Chuck Norris? Who
win?');
```

Properties

Properties of class

- Properties can be initialized in class

- Properties have access scope
  - public
  - protected
  - private

- Class can have constants

- Properties can be static

Normal properties and scope

- Normal properties are object level only
- Public access : everywhere
- Protected access : inside object and child type
- Private access : inside object

Normal properties and scope

```
class Human
{
    public $color;
    protected $mindSet = ['eat', 'sleep', 'reproduce'];
    private $brain = 'medium';

    public function setColor(string $color) {
        $this->color = $color;
        return $this;
    }

    public function teachTo(Human $student, string
$skill) {
        $student->learn($skill);
    }

    public function learn(string $skill)
    {
        if ($skill == 'COBOL') {
            $this->brain = 'damaged';
            return;
        }
        $this->teachTo($this, $skill);
    }
}
```

```
$student = new Human();
$teacher = new Human();

$teacher->setColor('green');
$teacher->teachTo($student,
'COBOL');
```

Time for exercise

```
git checkout OOP_exo_1
```

Scope

Class kind of scope

- Object scope
  - access properties and methods with **$this**
  - able to access static scope with **self**

- Static scope
  - trying to access object lead errors
  - access static properties and static methods with **self**
  - Owned by the class itself

Object scope example

```
class Human
{
    public $color;

    public function setColor(string
$color) {
        $this->color = $color;
        return $this;
    }
}

$dude = new Human();
$dude->setColor('magenta');
```

Normal property : object scope

Current object reference

New object

Normal method : object scope

Static scope

- Indicated by reserved word **static**
- Applicable to properties and methods
- Constants are always static
- Access via **self**

Static scope example

```
class Human
{
    public const MIND_SET = [
        'eat', 'sleep', 'reproduce', 'programing', 'driving'
...
    ];

    private $mindSet = ['eat', 'sleep', 'reproduce'];

    protected static $usedMindSet = [];

    public static function getMindset() {
        return self::MIND_SET;
    }

    public function addMindSet(string $skill) {
        if (in_array(self::MIND_SET, $skill)) {
            array_push($this->mindSet, $skill);
            array_push(self::$usedMindSet, $skill);
        }
    }
}

$dude = new Human();
$dude->addMindSet(Human::MIND_SET[10]);
$dude->addMindSet(Human::getMindset()[21]);
isset($dude::MIND_SET[32]);
```

constant : static scope

Normal property : object scope

Static property : static scope

Static access to constant in static context

Static access to constant in object context

Static access to static property in object context
(Note the leading $)

Static access to static class elements from outside

Object access to static class elements from outside

Life cycle

Class life cycle

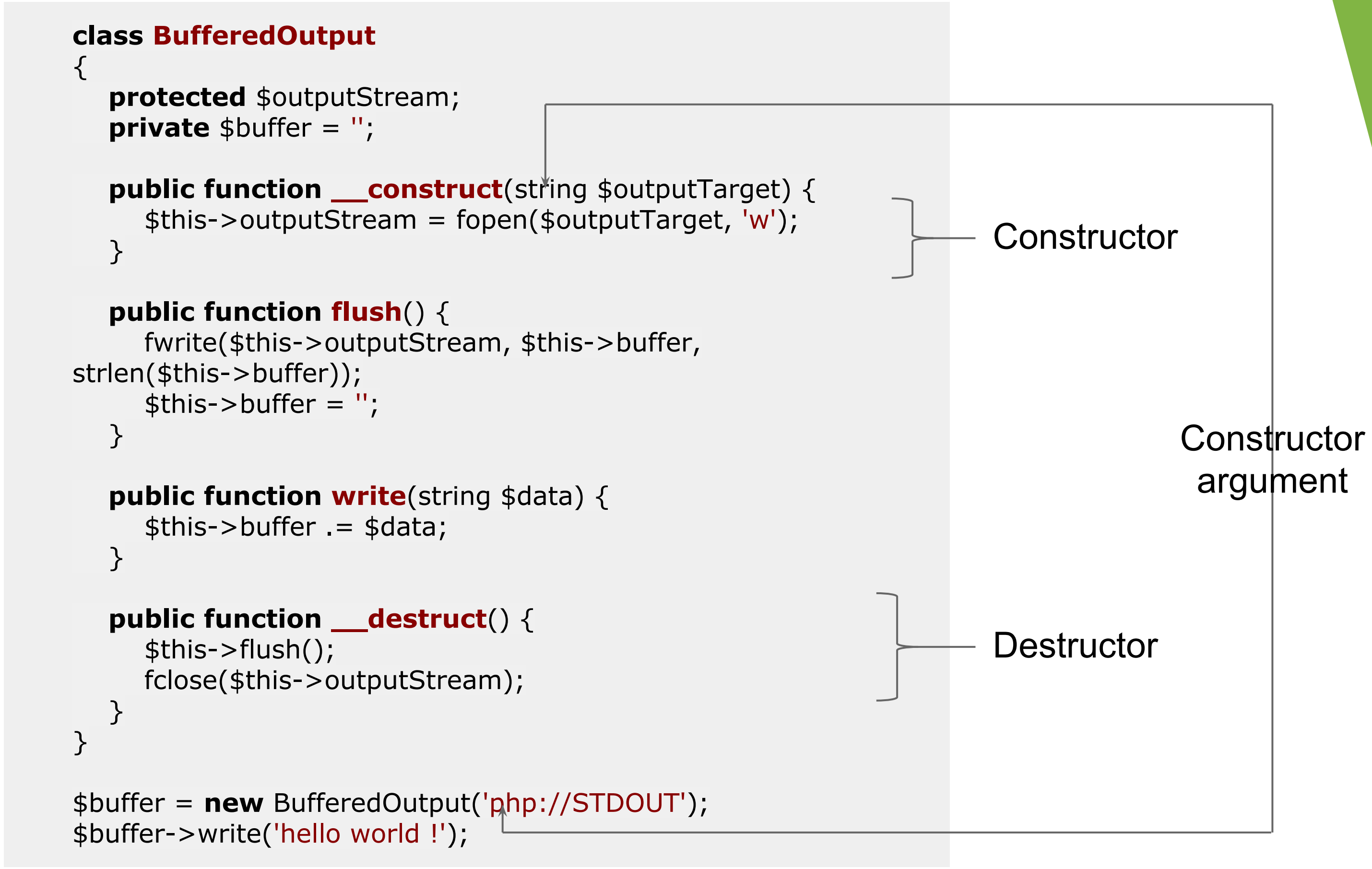
- Class have two main life event
  - Creation
  - Destruction

- Code can be plugged with “Magic Methods”

The life cycle magic methods

- At instantiation call to “__construct()”
    - Able to receive arguments
    - Never return value

- At deletion call to “__destruct()”
    - Not able to receive arguments
    - Fatal error in case of exception



```
class BufferedOutput
{
    protected $outputStream;
    private $buffer = '';

    public function __construct(string $outputTarget) {
        $this->outputStream = fopen($outputTarget, 'w');
    }

    public function flush() {
        fwrite($this->outputStream, $this->buffer,
strlen($this->buffer));
        $this->buffer = '';
    }

    public function write(string $data) {
        $this->buffer .= $data;
    }

    public function __destruct() {
        $this->flush();
        fclose($this->outputStream);
    }
}

$buffer = new BufferedOutput('php://STDOUT');
$buffer->write('hello world !');
```

Time for exercise

git checkout OOP_exo_2

Autoloading

Autoloading

- Standard PHP library feature
  - Use spl_autoload_register function

- Based on namespace natural path

Autoloading

```
spl_autoload_register(
    function($className){
        $filename = sprintf('%s/src/%s.php', __DIR__, str_replace('\\', '/',
$className));

        if (is_file($filename)) {
            require_once $filename;
        }
    }
);
```

Interface and inheritance

Interface
Ascendant inheritance
Abstract

Interface

Class

	Allowed to
define constant	TRUE
define properties	TRUE
implement methods	TRUE
implement interfaces	TRUE
extends another class	TRUE
be instantiated	TRUE
define methods to implement	FALSE
define abstract methods	FALSE

Interface

	Allowed to
define constant	TRUE
define properties	FALSE
implement methods	FALSE
implement interfaces	TRUE
extends another class	FALSE
be instantiated	FALSE
define methods to implement	TRUE
define abstract methods	FALSE

Interface

Extends other interfaces

```
interface QueueInterface extends \SplDoublyLinkedList,
\Serializable
{
    public const ARRAY_ASSOC = 0;
    public const ARRAY_NO_ASSOC = 1;

    public function toArray() : array;
}
```

Define constant

Define method to implement

WARNING

Only public constant and methods

In class

Implements interfaces

```
class Queue implements QueueInterface, \IteratorAggregate
{
    public function toArray() : array {
        return iterator_to_array($this->getIterator());
    }

    [...]
}
```

Ascendant inheritance

Ascendant inheritance

- Class can extend one and only one other class
- Class can override accessible methods
- Access to parent possible by using ‘**parent::**’

Ascendant inheritance

```php
class Animal
{
    private $color;
    protected $design;
    protected $cry;

    protected function cry() {
        return $this->cry;
    }
}

class Duck extends Animal
{
    public function __construct() {
        $this->cry = 'Quack';
        $this->design = 'Duck.png';
    }

    protected function cry() {
        return 'Quack! Quack!' . parent::cry();
    }
}
```

Duck inherit of Animal

Access to protected/public properties

Method override

Parent method call

Abstract

Abstract class

	Allowed to
define constant	TRUE
define properties	TRUE
implement methods	TRUE
implement interfaces	TRUE
extends another class	TRUE
be instantiated	FALSE
define methods to implement	FALSE
define abstract methods	TRUE

Abstract class

```
abstract class AbstractAnimal
{
    private $color;
    protected $design;
    protected $cry;

    protected abstract function cry();
}

class Duck extends AbstractAnimal
{
    public function __construct() {
        $this->cry = 'Quack';
        $this->design = 'Duck.png';
    }

    protected function cry() {
        return 'Quack! Quack!';
    }
}
```

Time for exercise

```
git checkout OOP_exo_3
```

Composer and Libraries

Composer

- Package manager for PHP
- Linked to packagist.org
- Manage package version
- Use JSON format to configure

Download

- getcomposer.org
- Pure PHP download process

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

php -r "if (hash_file('SHA384', 'composer-setup.php') ===
'544e09ee996cdf60ece3804abc52599c22b1f40f4323403c44d44fdfdd586475c
a9813a858088ffbc1f233e9b180f061') { echo 'Installer verified'; } else { echo
'Installer corrupt'; unlink('composer-setup.php'); } echo
PHP_EOL;"

php composer-setup.php

php -r "unlink('composer-setup.php');"
```

Composer init

- Initialize a package
  - Define the package name
  - A package description
  - Authors
  - Stability
  - Package type
  - License
  - Dependencies

Composer install

- Install the project dependencies
- Offer to install specific package
- Based on composer.lock

Composer update

- Update the project dependencies
- Offer to update specific package

Composer autoloading

- Automatically generated by install/update
- Custom repositories must be specified
    - key “autoload”
        - psr-4 array style
            - Key as namespace
            - value as folder

```
"autoload": {
    "psr-4": {
        "": "",
        "Model\\": "Model/",
        "Exception\\": "Exception/"
    }
}
```

Time for exercise

```
git checkout OOP_exo_4
```

Solid principle

S	Single responsibility principle
O	Open/closed principle
L	Liskov substitution principle
I	Interface segregation principle
D	Dependency inversion principle

OO pattern/principle	FP pattern/principle
Single Responsibility Principle	Functions
Open/Closed principle	Functions
Dependency Inversion Principle	Functions, also
Interface Segregation Principle	Functions
Factory pattern	Yes, functions
Strategy pattern	Oh my, functions again!
Decorator pattern	Functions
Visitor pattern	Functions